Acceleration of porous media simulations on the Cray XE6 platform

Kirsten M. Fagnan, Michael Lijewski, George Pau, Nicholas J. Wright
Lawrence Berkeley National Laboratory
1 Cyclotron Road
Berkeley, CA 94720

May 18, 2011

1 Introduction

In this paper we investigate the performance of the Porous Media with Adaptive Mesh Refinment (PMAMR) code which was developed in the Center for Computational Science and Engineering at Lawrence Berkeley National Laboratory. This code is being used to model carbon sequestration and contaminant transport as part of the Advanced Simulation Capability for Environmental Management (ASCEM) project. The goal of the ASCEM project is to better understand and quantify flow and contaminant transport behavior in complex geological systems. It will also address the long-term performance of engineered components including cementitious materials in nuclear waste disposal facilities, in order to reduce uncertainties and risks associated with DOE EM's environmental cleanup and closure activities. For this paper we have chosen a specific, computationally-intensive problem involving uranium contamination at a site in Georgia. Currently we are interested in how long it will take to simulate 25 years of flow, because this roughly corresponds to the time between experimental measurements; for example, see the measurements taken in Figure 1. Eventually, we would like to be able to simulate to 100 years or more, but this is likely to require the development of new algorithms beyond simple code tuning.

The PMAMR code is built upon the BoxLib framework and was previously parallelized using MPI [4]. In this work we used tools on the XE6 platform to identify performance bottlenecks and show how we can simulate 25 years of data faster by adopting a hybrid MPI/OpenMP programming model. Overall our hybrid approach yields a net speedup of 2.6× compared to the MPI only version, with substantially reduced memory usage. We note that while the use of OpenMP and MPI leads to considerable speed-ups, the calculations would still take on the order of a year to complete. Thus increasing the performance of our code with a hybrid programming model on Hopper is a first step towards the goal of simulating 25 years of flow.

1.1 Problem of Interest

The F-Area Seepage Basins are part of the Savannah River Site, a nuclear waste facility operated by the Department of Energy. The basins received process waste waters and stormwater runoff from the F-Area tritium and plutonium separation facilities from 1955 until 1988 [6]. The basins were designed to provide a repository where was could be stored in order to protect the surrounding environment. Over time, both normal operation and unusual events resulted in the addition of large amounts of NaOH and HNO_3 to the basins causing fluctuations in basin pH ranging in value from 0.6 to 13.2 [3]. This caused land in the vicinity of these basins to become acidic with elevated levels of metals, radionuclides, and nitrate. Waste deposition has significantly altered soils down gradient from the basins. One goal of the ASCEM project

is to determine reasonable ways to clean up theses sites, as well as understand how long it will be before the contaminated water will reach rivers and streams.

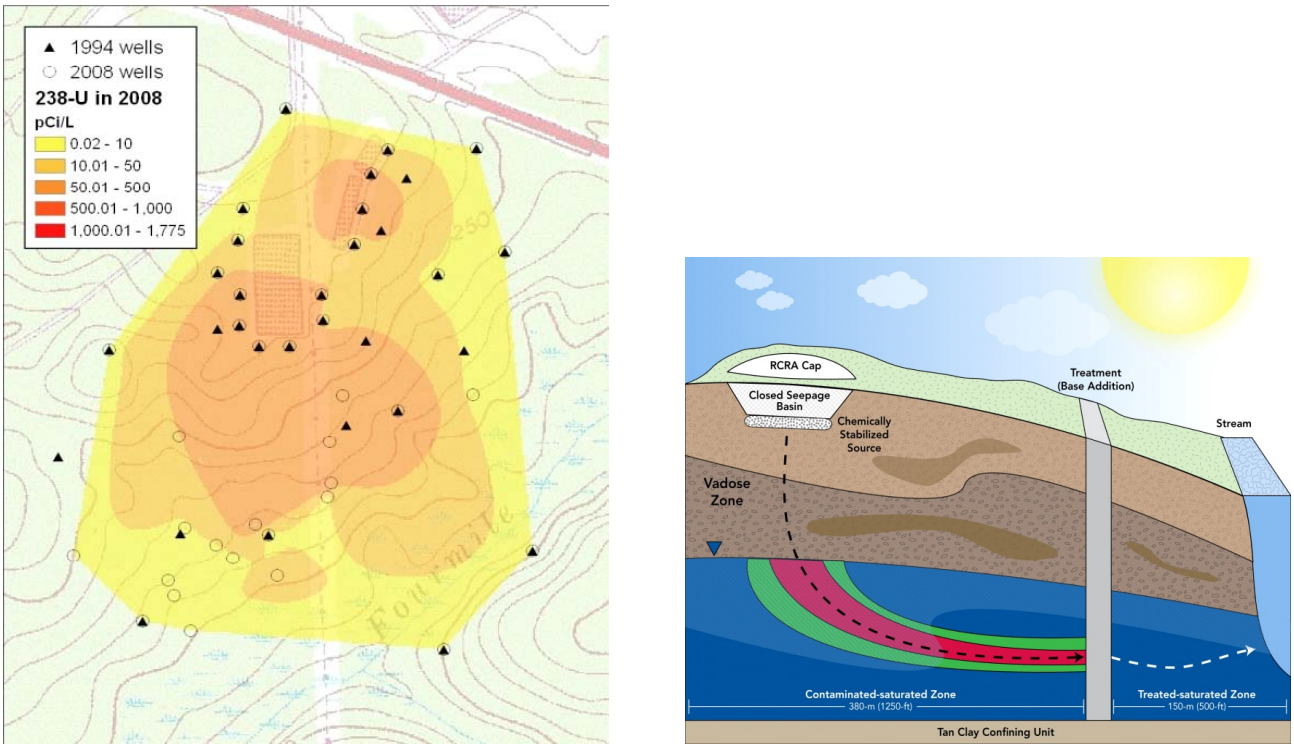


Figure 1: Cartoon of the F-basin subsurface contamination problem.

Figure 1 illustrates a proposed solution to the contamination problem, as well as the general geometry of the problem we are interested in modeling. The largest length scale along the horizontal axis is on the order of a kilometer, but the impermeable layers, vados zone and basine are all on the order of meters. The interfaces between layers in the soil where permeabilities (a measure of how easily fluid can pass through a material) vary by orders of magnitude, also need to be well-resolved in order for the multigrid solver to converge. These disparate spatial scales necessitate the use of adaptive mesh refinement (AMR) when using Cartesian or structured grids.

For this paper we model an idealized version of the F-basin problem with full chemistry and varying material properties that represent each of the different soil types and corresponding permeabilities. In Figure 2 we show the progression of fluid flowing from the top of the domain into a box that simulates the basin containing the chemicals. If we assume this causes the box to leak, then the flow will begin to permeate the domain. Once the influx is turned off, the flow is driven by gravity until it reaches the saturated groundwater zone. As the flow spreads into the domain, chemicals are reacting in the flow as well as in the surrounding material.

2 Equations of Porous Media

In order to better understand the algorithmic and computational approach taken in the PMAMR code, we provide a brief description of the equations being solved. We split the equations into hyperbolic, parabolic and elliptic parts and use appropriate solvers for each equation type. The parabolic and elliptic equations

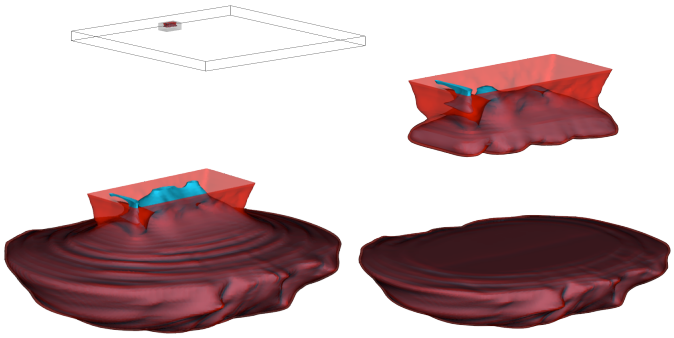


Figure 2: Results from a PMAMR calculation. a) Grid showing the entire domain illustrating the small size of the basin relative to the lengths in the $x-$ and $y-$directions. b) Fluid is infiltrating the domain from the surface (z =height), causing the chemicals to flow into the rest of the domain. c) and d) the flow of the chemicals progresses into more of the domain.

must be solved with an implicit method in order to avoid a severe time-step restriction. However, this is more difficult to parallelize because implicit methods require more communication between grids. The hyperbolic equation can be solved with an explicit method, that is relatively straightforward to parallelize.

2.1 Multiphase Multicomponent System

We solve the equations for multiphase (gas, liquid), multicomponent (air,water,oil) flow as formulated by G. Pau et. al. []. Given N_c components and N_p phases, the component density of the components, $\mathbf{n} \equiv \{n_i, 1 \leq i \leq N_c\}$, satisfies

$$\phi \frac{\partial n_i}{\partial t} + \nabla \cdot \frac{n_i}{s_{\mathcal{G}(i)}} \mathbf{v}_{\mathcal{G}(i)}(\mathbf{s}) = \nabla \cdot \phi \rho D \nabla X_i, \quad 1 \leq i \leq N_c, \tag{1}$$

where $\mathcal{G} : i \to \alpha$ maps component i to phase α — we consider only one-to-one mapping; $\mathbf{s} \equiv \{s_\alpha, 1 \leq \alpha \leq N_p\}$ and

$$s_\alpha = \sum_{i \in \mathcal{G}^{-1}(\alpha)} \frac{n_i}{\rho_i}; \tag{2}$$

ρ_i is the density of the component i; and $\mathbf{v}_\alpha \equiv \{v_{x,\alpha}, v_{y,\alpha}, v_{z,\alpha}\}$ is the velocity of the phase α and is defined by

$$\mathbf{v}_\alpha(\mathbf{s}) = -\lambda_\alpha(\mathbf{s})(\nabla p_\alpha(\mathbf{s}) - \rho_\alpha^p(n_{i \in \mathcal{G}^{-1}(\alpha)}; \rho_{i \in \mathcal{G}^{-1}(\alpha)})\mathbf{g}). \tag{3}$$

Here, $p_\alpha = p + p_{c,\alpha}$, ρ_α^p is the density of the phase α and λ_α is defined as

$$\lambda_\alpha(\mathbf{s}) = \frac{\kappa k_{r,\alpha}}{\mu_\alpha^p(n_{i \in \mathcal{G}^{-1}(\alpha)}; \mu_{i \in \mathcal{G}^{-1}(\alpha)})} \tag{4}$$

where μ_α^p is the viscosity of the phases; and $k_{r,\alpha}$ is the relative permeability for a given phase (Refer to subsequent section for more details). Note ρ_α^p and μ_α^p are in fact functions of the density, viscosity and partial density of the components in phase α.

We assume the density of phases does not change with composition and pressure. Then, the total velocity

$$\mathbf{v}_T = \sum_{\alpha=1}^{N_p} \mathbf{v}_\alpha \tag{5}$$

is divergence-free. Therefore,

$$\nabla \cdot \sum_{\alpha=1}^{N_p} \mathbf{v}_\alpha = \nabla \cdot \left(\sum_{\alpha=1}^{N_p} \lambda_\alpha(\mathbf{s}) \nabla (p(\mathbf{s}) + p_{c,\alpha}(\mathbf{s})) - \sum_{\alpha=1}^{N_p} \lambda_\alpha(\mathbf{s}) \rho_\alpha^p \mathbf{g} \right) = 0. \tag{6}$$

The phase velocity $\mathbf{v}_\alpha$ is then given by

$$\mathbf{v}_\alpha(\mathbf{s}) = \frac{\lambda_\alpha(\mathbf{s})}{\lambda_T(\mathbf{s})} \left\{ \mathbf{v}_T - \sum_{\beta=1,\beta\neq\alpha}^{N_p} \lambda_\beta(\mathbf{s})(\rho_\beta^p - \rho_\alpha^p)\mathbf{g} \right\} - \lambda_\alpha(\mathbf{s}) \left(\nabla p_{c,\alpha} - \sum_\beta \frac{\lambda_\beta(\mathbf{s}) \nabla p_{c,\beta}}{\lambda_T(\mathbf{s})} \right), \tag{7}$$

where $\lambda_T = \sum_\beta^{N_p} \lambda_\beta(\mathbf{s})$.

We now write (1) in the conservative form:

$$\phi \frac{\partial \mathbf{n}}{\partial t} + \nabla \cdot \mathcal{F}(\mathbf{n}, \mathbf{v}_T) = \nabla \cdot \phi \rho D \nabla \mathbf{X} + \nabla \cdot H \nabla P_c, \tag{8}$$

where

$$\mathcal{F}(\mathbf{n}, \mathbf{v}_T) = \sum_\alpha \frac{\mathbf{n}_\alpha}{u_\alpha} \left[\frac{\lambda_\alpha(\mathbf{s})}{\lambda_T(\mathbf{s})} \mathbf{v}_T + \lambda_\alpha(\mathbf{s}) \left(\rho_\alpha - \sum_\beta \frac{\lambda_\beta(\mathbf{s}) \rho_\beta}{\lambda_T(\mathbf{s})} \right) \mathbf{g} \right]; \tag{9}$$

$$H \nabla P_c = \sum_\alpha \left(\nabla p_{c,\alpha} - \sum_\beta \frac{\lambda_\beta(\mathbf{s}) \nabla p_{c,\beta}}{\lambda_T(\mathbf{s})} \right). \tag{10}$$

The governing equations are then given by Eq. 8 and 6. The nonlinear flux function (Eq. 9) necessitates further divisions into several subsystems.

2.2 Tracer Transport

The chemical species are treated as tracers being carried along with the fluid flow, we use the concentration of tracers as the primary variables. We assume the concentrations of tracers are small and do not affect flow behavior and there is no transfer of tracers between phases. Let C_j be the concentration of tracer j. The conservation equation of tracer j is then given by

$$\phi \frac{\partial s_i C_j}{\partial t} + \nabla \cdot (C_j \mathbf{v}_j) = \nabla \cdot \phi s_i D_j \nabla C_j; \tag{11}$$

D_j is the molecular diffusion (or product of tortuosity and molecular diffusion coefficient) and we do not consider dispersion.

3 Computational Implementation

In this section we discuss the algorithmic details involved in solving Eq. 8 and 6. We first give an overview of the basic time-stepping routine and then describe in greater detail each step of the algorithm, as well as the potential issues involved in parallelization of each step.

3.1 Temporal Discretization

The major steps of the algorithm for flow are:

Step 1: Solve the pressure equation at time t^m. Once p^m is known, we determine total velocity $\mathbf{v}^m$.

Step 2: Solve the component densities equation, advancing the solution from time t^m to time t^{m+1} using $\mathbf{v}^m$. The solution is denoted as $\mathbf{n}^{m+1,*}$.

Step 3: Using $\mathbf{n}^{m+1,*}$, we solve the pressure equation at time t^{m+1}. Once p^{m+1} is known, we determine total velocity $\mathbf{v}^{m+1}$.

Step 4: Defining $\mathbf{v}^{m+1/2} = \frac{1}{2}(\mathbf{v}^m + \mathbf{v}^{m+1})$, we solve the component densities equation again, advancing the solution from time t^m to time t^{m+1} based on $\mathbf{v}^{m+1/2}$. This step gives us the solution n^{m+1}.

3.2 Multigrid Solver for Elliptic Equations

The multigrid algorithm is a fast and efficient method for solving a wide class of integral and partial differential equations. The algorithm requires a series of problems be solved (in some sense) on a hierarchy of grids with different mesh sizes. For many problems, it is possible to prove that its execution time is asymptotically optimal. The niche of multigrid algorithms is large-scale problems where this asymptotic performance is critical, making it suitable for the problems we are investigating with PMAMR.

In the multigrid algorithm, it is desirable that the coarse and fine mesh partitions match in some way so that inter-processor communication during grid transfers is minimized. This is done by deriving coarse mesh partitions from the fine mesh partition. For example, when the coarse mesh points are a subset of fine mesh points, it is natural to simply use the fine mesh partitioning on the coarse mesh.

In BoxLib, the coarse and fine grids are partitioned together in an effort to minimize communication. There are limits that can be set by the user to determine both the maximum and minimum grid sizes used in the computation. Each time the domain is regridded, the grids are also redistributed to the different processors. The prior grid layouts are cached in an effort to speed up this process.

3.3 Time stepping

If we solved our equations using a fully implicit, coupled method, instead of operator splitting, we would be able to take much larger time-steps. Preliminary studies by collaborators in the ASCEM project indicate that a time step of .25 years is possible when using a fully implicit method, but the operator splitting approach limits the time step to be on the order of 10^{-4} years, which requires on the order of a million computational time steps to reach 100 years. We are interested in using the operator splitting type approach because these implicit methods introduce too much diffusion, which produces in the model large rates of flow of the contaminants that are non-physical. Since we are interested in chemical concentrations present in soil and groundwater, as well as when these contaminants will reach rivers and streams, it is important that the front of the chemical flow be represented as accurately as possible. The first step of our approach is to speed up the current code through the use of hybrid parallelization models. In later work we intent to develop an algorithm that would enable our code to take much larger time steps when the problem is in a quasi-steady state.

3.4 Chemical Reactions

The chemical equations in our model are solved using the geochemistry software developed by the AMANZI team at Lawrence Berkeley National Laboratory. The explicit discretization of the nonlinear reaction

equations are solved using an iterative Newton scheme. The AMANZI solver is called on each grid point. The local and explicit nature of the solver means that this part of our algorithm is embarrassingly parallel, making it a natural target for OpenMP threading.

3.5 BoxLib Data Structures

BoxLib contains the most fundamental objects used to construct a structured grid and its associated data structure. The objects in BoxLib are designed to support both uniform grid and structured grid adaptive mesh refinement (AMR), i.e. in a single calculation different regions of the domain can have different spatial resolutions. At each level of refinement, the region covered by that level is broken into boxes, or grids. The entire computational domain is covered by the coarsest (base) level of refinement, often called level 0. Higher levels of refinement have finer zones by a "refinement ratio" of either 2 or 4. Only a portion of the domain may be covered by the higher levels of refinement. As noted in Section 1.1, AMR is critical for the accurate resolution of the disparate spatial scales in the F-basin problem.

For parallel computations, the boxes are distributed to processors, in a fashion designed to put roughly equal amounts of work on each processor. We assume that the work done on each grid cell is proportional to the volume of that grid cell. Then load balancing is accomplished through the use of a Knapsack algorithm and space-filling curve, see Figure 3. This methodology works well when there is not a lot of spatial variance in the workload of the calculation. However, it is preferable to reduce the number of MPI tasks present in the problem, as this makes load balancing easier, which is another advantage to using MPI and OpenMP.

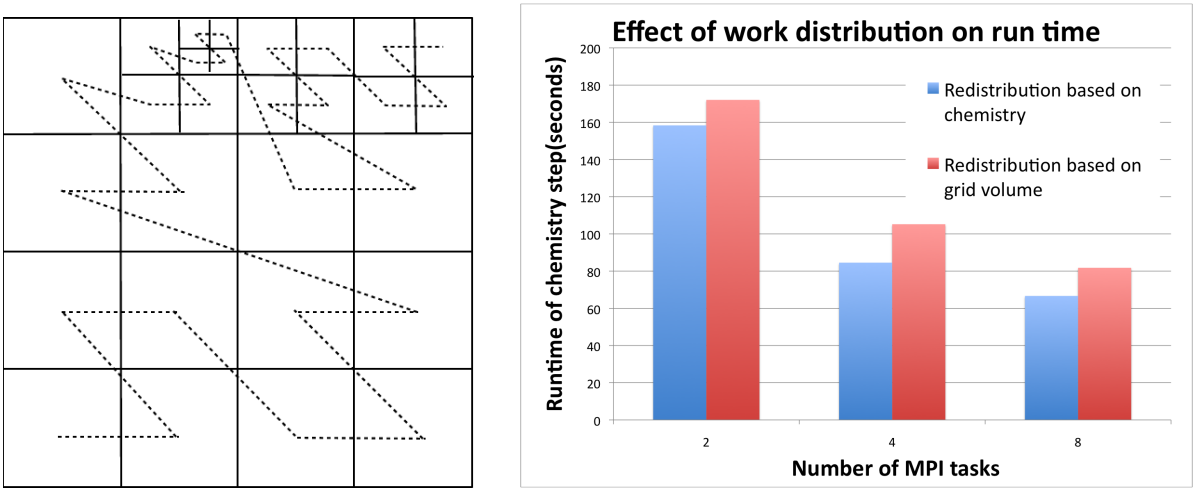


Figure 3: a) A cartoon illustrating the behavior of a space-filling curve that gets used when the amount of work per grid cell can be distributed by volume. b) The difference in the time to complete the chemistry step when work is distributed as a function of how difficult the chemistry solve is in a grid cell.

In calculations that involve intense chemistry that varies spatially, we redistribute the grids according to how much work each chemistry solve requires. This is done by keeping track of how hard the chemistry solve was in a grid cell at the previous time step. For example, in a nonlinear chemistry solve that relies on an iterative Newton method, we save the number of iterations required for convergence and use this as our metric for distribution of the MPI-tasks.

Figure 3 illustrates that as the calculation progresses and more chemistry is being done, distribution based on Newton iterations or some function count at the previous time step improves the performance of

our code. This is important because we need to simulate out to the thousands of time steps where the full 35-species system will be non-zero.

4 Hybrid Programming Model and Performance Results

Hopper, a Cray XE6, consists of 24-core compute nodes that are connected by a high-speed network. It is therefore logical to use OpenMP threading to exploit the multiple cores on each node, while using MPI to communicate between the compute nodes. One thing to note about Hopper is that, while there are 24 cores per node, the architecture is such that it makes sense to run one MPI task with 6 threads per 6-core NUMA node, as opposed to one MPI task with 24 threads. This leads to optimal performance on the compute node and avoids NUMA effects.

The performance of the pure-MPI version of PMAMR is limited by the chemistry solver. The time spent in the chemistry solve is 40% of the total calculation time. The solution to the chemical reaction equations is local in space, the solution at one grid point does not depend on the solution at neighboring grid points, thus it is a natural target for OpenMP parallelization. The solvers in PMAMR were developed using BoxLib have already been threaded and the performance of other codes (CASTRO and MAESTRO) based upon this approach has been investigated [1, 2]. Once we threaded the call to the chemistry solver, we discovered that the chemistry code had not been tested or optimized at all. The code was passing arrays by value instead of by reference, and it used many exception calls. These issues are known to cause performance issues when combined with threads. In fact, our initial results demonstrated that threading actually took twice as long as the serial code. Once these issues were fixed, the non-threaded chemistry still took 20% of the time for the problem we ran for 10 time steps. As the calculation continues out to a longer time, there is more complicated chemistry present, and the 20% value will increase.

In order to compare the performance the MPI-only versus the hybrid programming model, we ran a series of experiments where the number of compute nodes used was successively increased by a factor of 2. For each group of compute nodes, we ran an MPI-only job and 3 more jobs where we varied the number of threads. For example, on a single compute node, we ran job one with 24 MPI tasks, job 2 with 12 MPI tasks and 2 threads per task, job 3 with 8 MPI tasks and 3 threads per task and job 4 with 4 MPI tasks and 6 threads per task. We found that the mixed model that gave the best performance was the 4 MPI tasks with 6 threads per task, the same as was determined in [5]. We minimized the effect of job distribution across Hopper by running each experiment with the same `mppwidth`, or concurrency, on the same group of nodes.

In Figure 4, we plot the performance of the MPI-only and the hybrid programming model with 4 MPI-tasks and 6 OMP threads, for 10 computational steps of the PMAMR code. The MPI-only and hybrid model perform comparably up to 128 nodes, then the performance of the MPI-only model starts to decline. At this point the hybrid model out-performs the MPI-only model, by a factor of 1.6. On 256 nodes the hybrid approach is performing even better, 2.6× better than the MPI-only version.

The scaling of the hybrid version is still deviating from ideal scaling though. This is caused by both not having a large enough problem size to be able to chop up the grids further for distribution over more nodes, as well as not running the problem out to a long enough time where the performance of the chemistry has a greater influence. We are limited in how small a grid we can send to the multigrid solver. The nature of that algorithm requires coarsening of the grid cells and if we pass a grid that is too small to the solver, it can not be coarsened and the solver will fail.

It is our contention that if we were running out to longer times where more chemistry was being done in a larger part of the domain, as we would expect in the simulations out to several months or years, the hybrid model would perform even better. If we consider the plots in Figure 4, we can see that the time for the chemistry solve is scaling almost perfectly. As the flow progresses and chemistry is occurring in

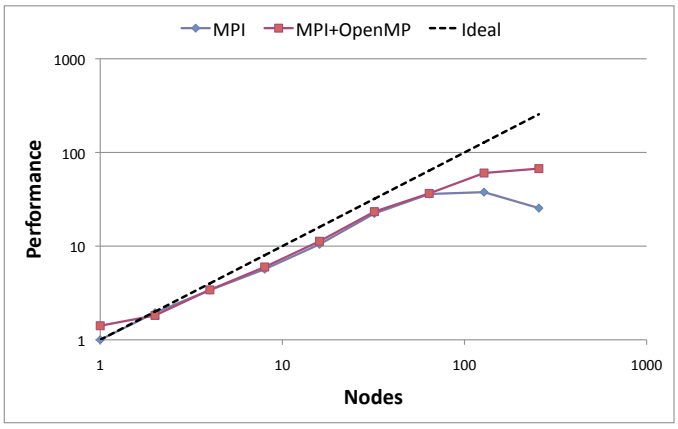
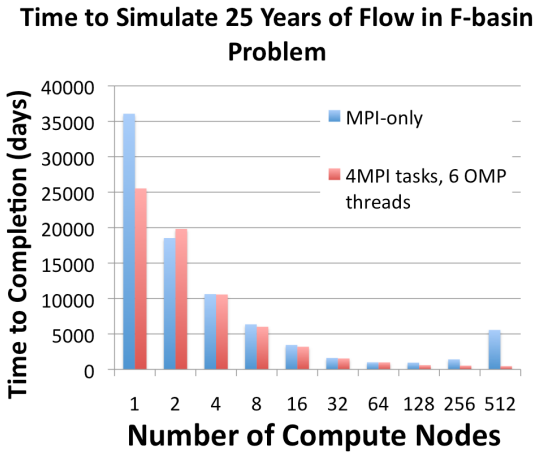
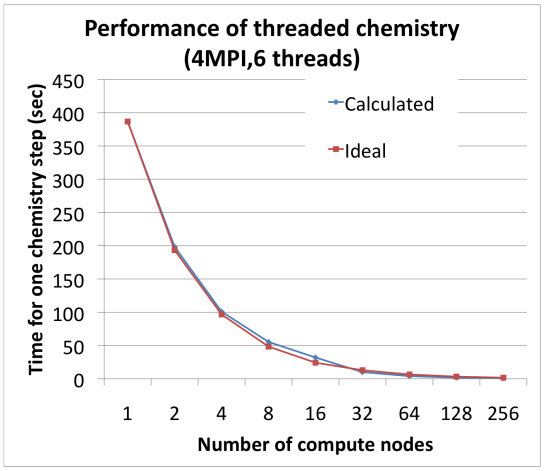


Figure 4: Performance of the MPI-only vs Hybrid model as a function of the number of nodes. As the number of nodes increases beyond 64, the Hybrid model performs better than the MPI-only model. These calculations were run on the Cray Hopper XE6 machine





a larger part of the spatial domain, the performance of the hybrid code should improve. At later times, the more of the species in the system of reactions will be non-zero, leading to more work in the chemistry solve, as observed in Figure 3. Distribution of the MPI tasks as a function of how hard the chemistry solve is, combined with OpenMP for threading the chemistry solve, should lead to greater performance improvement.

One of the stated goals of this work was to simulate the flow from a basin for 25 years. In an ideal world, with perfect scaling of our problem and time steps on the order of 300 seconds on the finest grid, on 49,152 processors or 2048 nodes on Hopper, we could run this calculation in 17 days. Figure 4 a) shows the progress we have made toward this goal so far. If we ignore computational cost and only consider the average time per step from experiments with 10 time steps, Figure 4 a) shows that as we increase the number of compute nodes, the hybrid model will solve the problem faster than MPI alone. The best average time per step obtained on 256 nodes will enable us to solve the 25 year problem in around 500 days. For MPI alone it would take more than 1400 days to complete. An increase in speed of almost 3×! Thus the hybrid model looks to be the more promising approach, and with additional work, we expect to further improve its performance.

5 Conclusions and Future Work

In order to understand the possible rates of contaminant transport it is essential to create accurate and computationally tractable models. Ideally we would like to be able to simulate 25 years of reactant flow with reasonable computational effort. Previous work using an all implicit scheme predicts unphysical rapid rates of transport because there is too much numerical diffusion present and the advective component is only first-order accurate. In contrast our mixed implicit/explicit scheme is second-order and produces more physically realistic results albeit at increased computational cost.

In this work we report preliminary results from our work to increase the performance of the PMAMR code using a hybrid OpenMP-MPI programming model. Using this approach we were able to increase the performance by up to 2.6× compared to a flat-MPI approach.

This hybrid programming model is a sensible approach for achieving good performance on the next generation of high-performance computing systems like Hopper that have many cores per node and less memory per core than their predecessors. The PMAMR code is built on BoxLib, a legacy code that has been heavily used and optimized by the CCSE group at LBNL. The chemistry solver in the code was written by collaborators within the ASCEM project. We found that, once the chemistry code was optimized, the chemistry solves were taking roughly 20% of our computational time and were written in an embarrassingly parallel manner. Thus were able to use OpenMP straightforwardly to thread the calls to the chemistry solver. Once the code was threaded, the improvement in the performance of the chemistry solve scaled perfectly, but as we moved to running our jobs on more nodes, the multigrid solver became the bottleneck. We are currently considering different methods for speeding up this calculation, like performing the coarse grid solve on a single compute node in order to minimize some of the communication costs. This is an avenue for future work.

At this time, PMAMR is a code that is a work in progress and we were unable to test the performance with more than two levels of refinement. While it is harder to examine the weak scaling of the problem when we add levels of dynamic, adaptive refinement, it does provide enough work that justifies the use of a larger number of compute nodes. These are experiments that we will perform in the future after more components of the code have stabilized.

If we use our metric of the time to complete a 25 year simulation of the F-basin problem, then the hybrid programming model is superior to MPI alone as the number of processors is increased. We can now do this calculation in approximately a year and a half, whereas with OpenMP, the best we could do was 2.5 years. The performance of the MPI-only code was significantly worse as the number of nodes was increased, but both models were far from ideal performance. Thus further work is needed. We are also pursuing an algorithmic approach to provide both the accuracy of the front of the chemical flow as well as the ability to take larger time steps. In the future, we would like to combine this algorithmic development with the hybrid programming model.

References

[1] A. Nonaka, et. al. Maestro: An adaptive low mach number hydrodynamics algorithm for stellar flows. *Astrophysical Journal Supplement Series*, 188:358–383, 2010.

[2] A. S. Almgren, et. al. Castro: A new compressible astrophysical solver. i. hydrodynamics and self-gravity. *Astrophysical Journal*, 715:1221–1238, 2010.

[3] J. W. Fenimore and J. H. Horton Jr. Operating history and environmental effects of seepage basins in chemical separations areas of the savannah river plant. *DPST-72-548, E. I. du Pont de Nemours and Co., Savannah River Laboratory, Aiken SC 29808.*, 1972.

[4] George S. H. Pau, et. al. "a parallel second-order adaptive mesh algorithm for reactive flow in geochemical systems". In *Proceedings of TOUGH Symposium, Berkeley, CA, USA*, September 2009.

[5] NERSC. Performance implications of using openmp, 2011. `http://www.nersc.gov/users/computational-systems/hopper/performance-and-optimization/#OpenMP\%20Performance`.

[6] S.M. Serkiz, et. al. Environmental availability of uranium in an acidic plume at the savannah river site. *Vadose Zone Journal*, 6:354–362, 2007.